Embedded.com

Solving Switch Bounce Problems

By Jack Ganssle, Courtesy of Embedded Systems Programming
Apr 21 2004 (13:00 PM)
URL: http://www.embedded.com/showArticle.jhtml?articleID=18902552

No chaste unblemished logic signals these; a scope will show the contacts torturously bouncing before settling into a stable state.

Click here for reader response to this article

As you read this, millions of contacts around the world are incessantly clacking. Relays, sensors, and switches—they're all ugly mechanical monstrosities that reflect the raw grittiness of the physical world. Contacts whack and drag across each other, their shuddering dance creating analog waveforms unique as snowflakes. No chaste unblemished logic signals these; a scope will show the contacts torturously hitting and rebounding, bouncing for milliseconds before finally settling into a stable state.

Last month I described an experiment I conducted that evaluated the bounce characteristics of a number of switches. The data showed bounce times varying from a microsecond to fractions of a second. Identical switches were wildly different. Some conductive elastomer devices never really switched, per se; their outputs oozed in a slow ramp from zero to a logic one. Others behaved in a way sure to confuse the typical debounce code we routinely employ in our firmware.

I bet you figured this month I would describe better algorithms, giving useful code snippets. Sorry! For debouncing is one of those activities that lies between the code and the hardware. In some cases a hardware cleanup is the right approach.

Set and forget

Figure 1 shows the classic debounce circuit. Two cross-coupled NAND gates form a simple Set-Reset (SR) latch. The design requires a double throw switch. Two pull-up resistors generate a logic one for the gates; the switch pulls one of the inputs to ground.

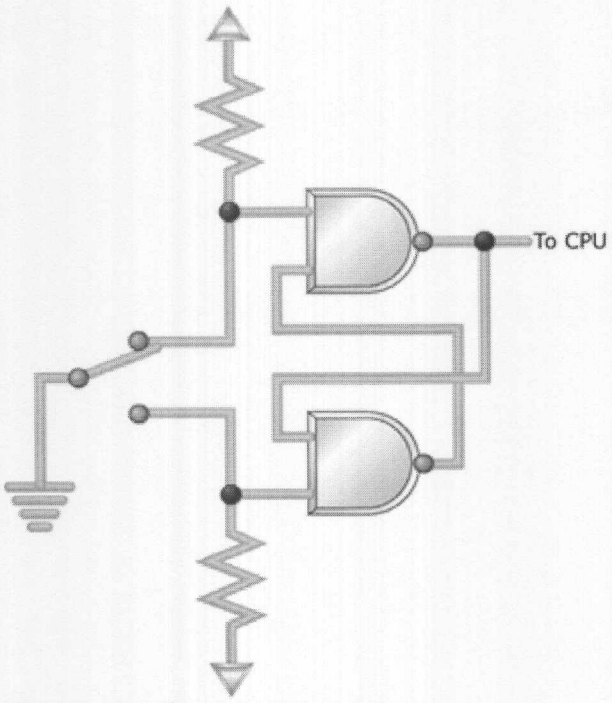

Figure 1: The SR debouncer

The SR latch is a rather funky beast, as confusing to non-EEs as recursion is to, well, just about everyone.

With the switch in the position shown the upper gate's output will be a one, regardless of the value of the other input. That and the one created by the bottom pull-up resistor drives the lower NAND to a zero, which races around back into the other gate. If the switch moves between contacts and is for a while suspended in the nether region between terminals, the latch maintains its state because of the looped-back zero from the bottom gate.

The switch moves a rather long way between contacts. It may bounce around a bit, but will never bang all the way back to the other contact. Thus, the latch's output is guaranteed bounce-free.

The circuit suggests an alternative approach, a software version of the same idea. Why not skip the NAND pair and run the two contacts, with pull-ups, directly to input pins on the CPU? Sure, the computer will see plenty of bounciness, but write a trivial bit of code that detects any assertion of either pole, which means the switch is in that position, as follows:

```
if(switch_hi())state=ON;
if(switch_lo())state=OFF;
```

The functions **switch_hi** and **switch_lo** each read one of the two poles. Other functions in the program examine variable **state** to determine the switch's position. This saves two gates but costs one extra input pin on the processor. It's the simplest—and most reliable—debounce code possible.

(OK, so I slipped some code in after all).

An RC debouncer

The SR circuit is the most effective of all debouncing approaches but it's rarely used. Double-throw switches are bulkier and more expensive than the simpler single-pole versions. An awful lot of us use switches that are plated onto the circuit board, and it's impossible to make double-pole versions of these. So EEs prefer alternative designs that work with cheap single-pole switches.

Though complex circuits using counters and smart logic satisfy our longing for pure digital solutions to all problems, from signal processing to divorce, it's easier and cheaper to exploit the peculiar nature of a resistor-capacitor (RC) network.

Charge or discharge a capacitor through a resistor and you'll find the voltage across the cap rises slowly; it doesn't snap to a new value like a sweet little logic circuit. Increase the value of either component and the time lag ("time constant" in EE lingo) increases.

Figure 2 shows a typical RC debouncer. A simple circuit, surely, yet one that hides a surprising amount of complexity.

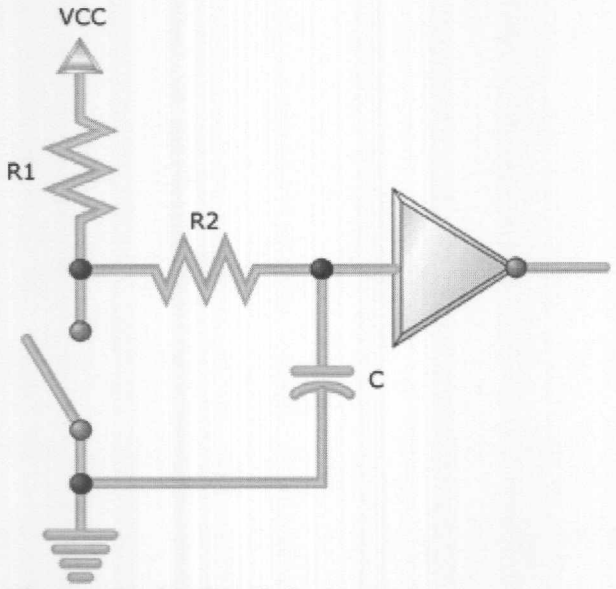

Figure 2: An RC debouncer

Suppose our fearless flipper opens the switch. The voltage across the cap is zero, but it starts to climb at a rate determined by the values of R_1, R_2, and C. Bouncing contacts pull the voltage down and slow the cap's charge accumulation. If we're very clever in selecting the values of the components the voltage stays below a gate's logic one level till all of the whacking and thudding ceases. (If the time constant is too long, of course, the system won't be responsive to fast switch actuations.)

The gate's output is thus a pristine bounce-free logic level.

Now suppose the switch has been open for a while. The cap is fully charged. Snap! The user closes the switch, which discharges the cap through R_2. Slowly, again, the voltage drools down and the gate continues to see a logic one at its input for a time. Perhaps the contacts open and close a bit during the bouncing. While open, even if only for short periods, the two resistors start to recharge the cap, reinforcing the logic one to the gate. Again, the clever designer selects component values that guarantee the gate sees a one until the clacking contacts settle.

Squalid taverns are filled with grizzled veterans of the bounce wars recounting their circuits and tales of battles in the analog trenches. Most will puzzle over R_2, and that's not entirely due to the effects of the cheap booze. The classic RC debouncer doesn't use this resistor, yet it's critically important to getting a thwack-free output from the gate.

R_2 serves no useful purpose when the switch opens. R_1 and C effectively remove those bounces.

But strange things can happen when suddenly discharging a capacitor. The early bouncing might be short, lasting microseconds or less. Though a dead short should instantly discharge the cap, there are no pristine conditions in the analog world. The switch has some resistance, as do the wires and printed circuit board traces that interconnect everything.

At high speeds every wire is actually a complex circuit. You wouldn't think a dull-headed customer flipping the switch a few times a second would be generating high-speed signals, but sub-microsecond bounces, which may have sharp rise times, have frequency components in the tens of megahertz or more. Inductance and stray capacitance raise the impedance (AC resistance) of the closed switch. The cap won't instantly discharge.

Worse, depending on the physical arrangement of the components, the input to the gate might go to a logic zero while the voltage across the cap is still one-ish. When the contacts bounce open the gate now sees a one. The output is a train of ones and zeroes—bounces.

R_2 ensures the cap discharges slowly, giving a clean logic level regardless of the storm of bounces.

Another trick lurks in the design. The inverter cannot be a standard logic gate. TTL, for instance, defines a zero as an input between 0.0 and 0.8V. A one starts at 2.0. In between is a demilitarized zone, which we're required to avoid. Feed 1.2V, for example, to such a gate and the output is unpredictable. But this is exactly what will happen as the cap charges and discharges.

Instead use a device with *Schmitt trigger inputs*. These devices have hysteresis; the inputs can dither yet the output remains in a stable, known state.

Never run the cap directly to the input on a microprocessor, or to pretty much any I/O device. Few of these have any input hysteresis.

Doing the math

The equation for discharging a cap is:

Equation 1:

$$V_{th} = V_{final} e^{-t/RC}$$

where:

- V_{cap} is the voltage across the capacitor at time t
- $V_{initial}$ is the voltage initially on the cap
- t is the time in seconds
- R and C are the values of the resistor and capacitor in ohms and farads, respectively

The trick is to select values that ensure the cap's voltage stays above V_{th}, the threshold at which the gate switches, till the switch stops bouncing. It's surprising how many of those derelicts hanging out at the waterfront bars pick an almost random time constant. "The boys 'n' me, we jest figger sumpin like 5ms." Shortchanging a real analysis starts even a clean-cut engineer down the slippery slope to the wastrel vagabond's life.

Most of the switches I examined last month had bounce times well under 10ms. Use 10 to be conservative. Now increase that by the bounce duty cycle. Thumping contacts will slow the capacitor's charge. My data shows we can expect about a 50% duty cycle, giving us 20ms.

Rearranging the time constant formula to solve for *R* (the cost and size of caps vary widely so it's best to select a value for *C* and then compute *R*) yields:

Equation 2:

$$R = \frac{-t}{C \ln\left(\frac{V_{th}}{V_{final}}\right)}$$

Though it's an ancient part, the 7414 hex inverter is a Schmitt trigger with great input hysteresis. The AHCT version has a worst-case V_{th} for a signal going low of 1.7V. Let's try 0.1µF for the capacitor since those are small and cheap and solve for the condition where the switch just closes. The cap discharges through R_2. If the power supply is 5V (so $V_{initial}$ is 5), then R_2 is 185kΩ. Of course, you can't actually *buy* that kind of resistor, so use 180kΩ.

But the analysis ignores the gate's input leakage current. A CMOS device like the 74AHCT14 dribbles about a microamp from the inputs. That 180kΩ resistor will bias the input up to 0.18V, uncomfortably close to the gate's best-case switching point of 0.5V. Change the cap to 1µF and use 18kΩ for R_2.

$R_1 + R_2$ controls the cap's charge time, and so sets the debounce period for the condition where the switch opens. The equation for charging is:

Equation 3:

$$V_{cap} = V_{initial}\left(1 - e^{\frac{-t}{RC}}\right)$$

Solving for *R*:

Equation 4:

$$R = \frac{-t}{C \ln\left(1 - \frac{V_{th}}{V_{initial}}\right)}$$

V_{final} is the final charged value—the 5V power supply. V_{th} is now the worst-case transition point for a high-going signal, which for our 74AHCT14 a peachy 0.9V. $R_1 + R_2$ works out to 101kΩ. Figure on 82kΩ for R_1.

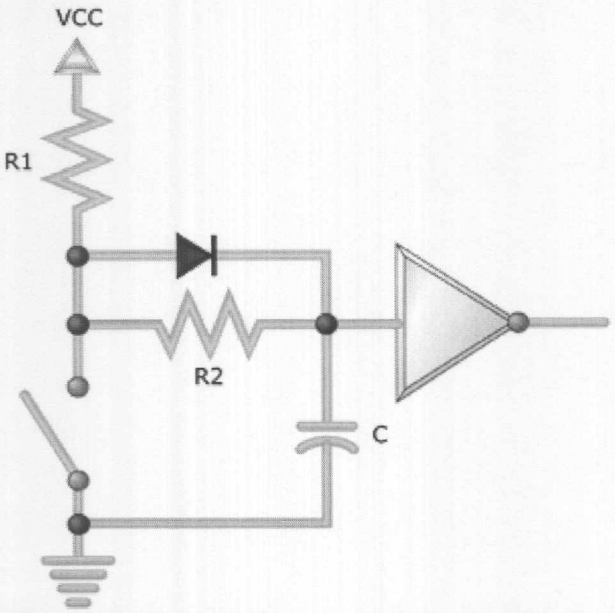

Figure 3: An RC debouncer that actually works

The diode is an optional part needed only when the math goes haywire. It's possible, with the wrong sort of gate where the hysteresis voltages assume other values, for the formulas to pop out a value for $R_1 + R_2$ which is less than that of R_2. The part forms a shortcut that removes R_2 from the charging circuit. All of the charge flows through R_1.

Equation 4 still applies, except we have to account for drop across the diode. Change V_{final} to 4.3V (5 minus the 0.7 diode drop), turn the crank and R_1 pops out.

Be wary of the components' tolerances. Standard resistors are usually ±5%. Capacitors vary wildly; +80/-20% is a common rating for electrolytics; Even small ceramics might vary by ±30%.

Other thoughts

Don't neglect to account for the closed resistance of oddball switches. Some conductive elastomer devices exceed 200Ω.

Two of the elastomer switches I examined last month didn't bounce at all; their output smoothly ramped from zero to +5V. The SR and RC debounce circuits are neither necessary nor effective in these cases. It's better to run the switch directly into a Schmitt trigger's input.

Never connect an undebounced switch to the clock of a flip-flop. The random bounce hash is sure to confuse the device. A 74HCT74 has a max rise and fall time spec of 6ns—easily exceeded by some of the 18 switches I tested.

The 74HC109 requires a minimum clock width of 100ns. I found pulses shorter than this in my experiments. Its higher-tech brother, the 74HFC109 actually has a Schmitt trigger clock input—it's a much safer part to use when connected to real-world events.

Similarly, don't tie undebounced switches, even if Schmitt triggered, to interrupt inputs on the CPU. Usually the interrupt pin goes to the clock input of an internal flip flop. As processors become more complex their datasheets give less useful electrical information; they're awash in programming data but leave designers adrift without complete timing specs. Generally we have no idea what the CPU expects as a max rise time or the min pulse width. Those internal flops aren't

perfect, so don't flirt with danger by feeding them garbage.

Next month I'll give an Embedded Systems Conference wrap-up. More on debouncing after that.

Jack G. Ganssle is a lecturer and consultant on embedded development issues. He conducts seminars on embedded systems and helps companies with their embedded challenges. Contact him at jack@ganssle.com. **Reader Response**

An often neglected debounce method that I have used successfully in a game is to sample the output of a switch with a pull-up at a period longer that the bounce duration.

Before the bounce the input is certainly in one state.

When sampled during the bounce the state may read in either state.

When the next sample is tacken the switch will have stopped bouncing in the other state.

The effect of the bounce is to jitter the detection of the state change by one sample period.

In many cases this is acceptable.

I suspect that this would not be acceptable in a really fast action "shooter" game, but in the original PDP-1 "Spacewar!", it worked well for many years with many switches.

- Steve Russell

Thanks for the response. My method is very similar to that of Steve Russell and works very reliably with small PCB tactile switches. I often hear of EEs going to extraordinary lengths to debounce switches when often this very simple method works well for a lot less effort.

I have also used the R1 (10k or less) + R2 (100k) + C (100nF) circuit (10ms time constant) straight into a CPU input that is sampled every 20ms to 50ms. I remember the previous state of the input. If I sample while the input level is illegal, it may sample low or high. If it is the same as before, no change = fine. If it samples different, it must be changing = fine. Next sample it will be stable. Don't use this method on clocked inputs!

Above all, use a technique that is appropriate for the switches and IC input in use. Fortunately, I don't see many elastomer contacts...

- Les Grant